

L
E
D
A



ASSOCIATION FRANCAISE DE GENIE LOGICIEL

La LETTRE n° 26

Janvier 1997



L'an 2000 : Peur ou opportunité ?

C'est inéluctable ; nous allons changer de millénaire.

Les puristes, qui vont encombrer les courriers des lecteurs, nous démontreront par 0 + 1 que nous ne changerons de millénaire que dans la nuit du 31 décembre 2000 au 1^{er} janvier 2001.

Cependant pour les ordinateurs, c'est la nuit du 31 décembre 1999 au 1^{er} janvier 2000 qui, aux dires des oracles, constitue la plus grande menace pour la continuité temporelle.

Un peu d'histoire

Certains historiens nous racontent que nos ancêtres ont connu la grande peur de l'an 1000. Pourtant, ils ne disposaient même pas du vieil almanach des P.T.T. qu'ils auraient été, pour la plupart, bien incapables de lire. Les bases de la numération étaient détenues par une élite qui les gardaient jalousement. Comment, dans ces conditions, une telle panique a-t-elle pu se répandre simultanément dans toutes nos provinces, chez tous ces paysans aux prises avec leurs difficultés habituelles (mauvaises récoltes, épidémies, conflits armés, etc.) ? C'est un premier mystère qui pourrait peut-être en expliquer un autre.

Vous vous souvenez, aussi, de la grande frousse soulevée par un certain vendredi 13 qui devait déclencher l'assaut concerté de tous les virus et provoquer l'évaporation de tous les écrans d'ordinateur, dans un sardonique éclat de rire.

Quel est le risque informatique de l'an 2000 ?

Au cours des débuts de l'ère informatique, dans les années 60, la mémoire rare coûtait fort cher ; on a pris la mauvaise habitude de représenter les dates en jours mois an : JJ/MM/AA.

Le jour JJ est compris entre 1 et 31 - ce qui est toujours valable ; le mois MM, entre 1 et 12 - ce qui est toujours valable. L'année AA était comprise entre 00 et 99 ce qui était valable de 1900 à 1999 ; période pendant laquelle les deux premiers chiffres 1 et 9 étaient inutiles.

Or, pour les périodes qui franchiront le 1^{er} janvier 2000, cette notation s'avère impropre à tout calcul reposant sur une comparaison de dates.

La sensibilisation

Les dirigeants d'entreprise sont interpellés sur l'imprévoyance de leur service informatique. La presse professionnelle entretient leur inquiétude en égrenant, semaine après semaine, les secondes qui restent avant la date fatidique. Le Gartner Group a avancé des chiffres impressionnants : l'équivalent de quelques milliards de dollars en hommes.jours à dépenser pour éviter le pire. Attention, plus on se rapprochera de la date critique, plus cela coûtera cher.

Face à un tel défi, il n'est plus question de prendre des décisions fondées sur des hypothèses de retour sur investissement. Ce grand nettoyage des dates est une opération qui ne rapportera strictement rien aux entreprises, et contrairement aux projets traditionnels, il n'y a aucune possibilité de différer la date de lancement, même si l'on ne s'estime pas suffisamment prêt !

Cet investissement évitera seulement les difficultés provoquées par une date fantaisiste qui déclencherait une cascade imprévisible de catastrophes.

D'ores et déjà, les sociétés informatiques proposent leurs services et les outils, forgés à cette occasion, pour soulager les entreprises de ce travail fastidieux.

Et si on en profitait ?

Cette grande peur peut cependant avoir un effet bénéfique : c'est l'occasion unique de faire un grand ménage dans le parc de ses logiciels, en les classant en trois familles.

- La première famille est celle des anciens programmes. Ils sont périmés, obsolètes, inadéquats, on ne sait plus très bien pourquoi et pour qui ils ont été écrits ; on a perdu la documentation, mais on les exploite par habitude et on les garde ... au cas où.
- La seconde famille est celle des applicatifs, conçus récemment pour répondre aux besoins actuels des utilisateurs, ils sont documentés et ils intègrent des dates complètes.
- La troisième famille constitue le domaine intermédiaire des applications qui ne sont pas encore tout à fait assez fatiguées pour être éliminées mais jugées insuffisamment fiables pour passer le siècle en l'état. Une analyse très fine, au cas par cas, s'impose.

Il faut éliminer sans pitié et sans regret tous les membres de la première famille.

La seconde ne pose aucun problème.

Quant à la troisième, un examen attentif de chaque système applicatif permettra de déterminer s'il faut s'en débarrasser, le conserver en l'état ou le rénover. La voie de la rénovation ouvre des perspectives variées qui recouvrent une gamme d'actions très différentes qui s'échelonnent :

- * de la simple extension automatique des dates (attention, il y a des dates cachées dans des profondeurs technologiques insoupçonnées)
- * à une nouvelle conception de tout ou partie de l'application, autour d'un progiciel ou d'un logiciel spécifique.

Bon courage, faites vite et bien !...Le calendrier n'attendra pas. ▲

Alain Coulon



L'an 2000 : Mythe ou réalité ?

À l'aube de 1997, quels vœux pouvons-nous adresser aux responsables informatiques ?

Comme chaque année sont attendues les étrennes généreusement octroyées par les Directions Générales. On y rencontre, pêle-mêle, les sempiternels présents, toujours désirés, rarement offerts : maintien du budget de fonctionnement de l'informatique, déblocage de fonds pour l'acquisition de logiciels, Mips qui redonneraient du cœur à l'ouvrage au mainframe exsangue...

Cette année, pourtant, il semble qu'un nouvel article figure sur la liste de Noël, pieusement remise au Directeur Général. Cet article n'est pas ludique, loin s'en faut. Il ne joint certainement pas non plus l'utile à l'agréable. Il est cependant nécessaire, même indispensable dans la panoplie du parfait DI (Directeur Informatique).

*Cessons là le suspens, et rentrons dans le vif du sujet ! Nous dissertons sur ce merveilleux présent tant attendu : un budget pour le projet de **migration An 2000**.*

Eh oui, j'en vois qui font la moue. Ils s'attendaient peut-être à quelque chose de plus drôle où il serait question de Client-Serveur (est-il toujours de mode sur la liste du Père Noël ?), de multimédia, voire d'Internet et même d'Intranet. Non, vous m'en voyez désolé, mais l'An2000 est le cadeau (empoisonné ?), que nos amis DI devront accepter et assumer dans les prochains mois.

Et si, avant de nous plonger dans le prochain millénaire, nous remontions un peu le temps...

Un bruit confus...

Il était une fois, un responsable informatique qui traçait des plans sur la comète (pour ne pas parler de châteaux en Espagne) quand aux évolutions futures de son système d'information.

Il rêvait aux nouvelles technologies, pensait Web, données décentralisées, outils de n^{ième} génération, multimédia.

Installé confortablement sur son siège (qui est quelquefois éjectable), il imaginait ce que serait l'informatique de l'an 2000. À n'en pas douter l'urbanisme permettrait des échanges de flux sécurisés et ouvrirait les portes de l'informatique universelle. Quel bonheur d'avoir la chance d'être celui qui conduirait l'entreprise vers ces nouveaux horizons informatiques !

Mais, brusquement une rumeur vient le distraire de son rêve. Il tend l'oreille. Qu'est ce bruit venu du fond des couloirs ? Il croit entendre des mots bizarres, un peu surannés dans le contexte où il était plongé quelques instants auparavant. Des mots reviennent, colportés par les médias : migration, an 2000, sinistres, blocage du système d'information, pertes irréversibles, désastre... Que signifie tout cela ?

Appelant son Directeur de la Production, un homme sur qui il peut s'appuyer, à qui on ne la fait pas ; il lui demande de se renseigner, lui faisant comprendre que de telles perturbations l'empêchent de se consacrer à sa tâche principale : prévoir ce que sera l'informatique de l'entreprise au début du 21^{ième} siècle.

Le Directeur de la Production a vécu déjà de nombreuses migrations ; il se souvient de l'époque où le DOS/VSE laissa la place à un MVS triomphant ; il se souvient que ses équipes (aux effectifs restreints dans le monde DOS) avaient mené promptement ce projet et qu'il n'y avait pas eu de conséquences néfastes sur le fonctionnement global de l'informatique.

Fort de cette expérience, il informe son DI de la nécessité de prévoir un projet de migration technique pour l'an 2000, projet peu coûteux au demeurant et qui pourra être mené en interne en quelques semaines, voire quelques mois...

Ouf, notre Directeur Informatique peut reprendre ses travaux de prospective ; il est rassuré.

Mais, bientôt, une nouvelle rumeur apparaît. Elle prend corps dans la presse, dans les salons, dans les séminaires, enfin dans tous les endroits où les acteurs du monde informatique se retrouvent habituellement.

De nouveaux mots apparaissent : analyse d'impact, lotissement, interfaçage, transformation des structures de données, tests de non-régression, mises en production successives, fonctionnement en double...

Que ces propos semblent disproportionnés pour un projet technique ne présentant pas, a priori, de difficultés majeures !

Une rapide prise de conscience

Un changement de comportement apparut alors. Les Directeurs Informatiques prirent conscience que ce projet n'était peut-être pas aussi simple qu'il y paraissait de prime abord. Notre DI décida alors de se faire sa propre opinion. On put le voir, durant quelques semaines, fréquenter assidûment les séminaires « An 2000 », dévorer avec empressement les articles de presse traitant du sujet, recevoir les SSII qui sans cesse l'appelaient pour l'alerter sur l'urgence de la situation.

Il lui fallut peu de temps pour découvrir l'ampleur du problème. En fait, il ne s'agissait, ni plus ni moins, que d'effectuer une maintenance touchant le système d'information dans sa globalité. Il comprit à ce moment que la tâche qui attendait ses équipes était bien plus importante que ce qu'il avait pu supposer dans un premier temps.

Mais rien n'était perdu, déjà de nouveaux **gourous** (l'informatique en voit naître et disparaître assez souvent...) apparaissaient ; ils promettaient que grâce à eux tout se passerait d'une façon transparente.

Il y avait diverses écoles. Il croisât les tenants des algorithmes miracles qui proposaient de régresser dans le temps, de changer la base de numérotation de nos dates. Il rencontra aussi ceux qui savaient automatiser le changement du code dans les programmes. Il fut tenté de les croire, il était si rassurant de pouvoir se dire qu'en fait l'An2000 était un faux problème inventé par une presse en mal de sensationnel (la multiplicité des revues informatiques induisant, il est vrai, la recherche du scoop pour vendre, mais ceci est un autre sujet...). Hélas il déchantait très vite, la problématique de l'An2000 existait bel et bien, ce n'était pas une mince affaire, le risque n'était pas illusoire, il fallait ouvrir **un grand chantier...**

Causes et conséquences

Quelle est donc la réalité de ce que l'on appelle le problème de l'An 2000 ?

L'énoncé du sujet est fort simple : nos systèmes d'information, remontant pour la plupart aux années soixante-dix, ont été conçus en gérant au mieux deux paramètres, qui à l'époque pesaient un poids considérable : la **rareté de la mémoire** et la **puissance limitée des machines**.

Tous ceux qui eurent à programmer à cette époque le savent bien, ces deux paramètres ont influencé la conception fonctionnelle et technique des applications. Il fallait faire avec ce dont nous disposions (tous les possesseurs de micros ignorent que la puissance de leur ordinateur personnel est des dizaines de fois plus importante que celle de nos vieux IBM 360 et 370).

Le résultat était la recherche perpétuelle du gain de place. Donc, lorsqu'il fallait stocker une date on allait au plus court : une année sur 2 caractères (78 signifiant 1978...). Ignorions-nous pour autant l'imminence du prochain millénaire ? Certes non, mais qui aurait pu croire que les applications que nous développons vivraient encore vingt ans plus tard ? Ne pensez pas non plus que nous agissions différemment à la fin des années 80, puisque la généralisation tant attendue (et que nous attendons toujours) du Client-Serveur impliquait, à court terme, la refonte du système d'information qui ne pourrait plus tourner sur le mainframe, puisque celui-ci aurait disparu...

Ainsi, la plupart des années composant les dates sont stockées et traitées sur 2 caractères. Or, en informatique de gestion la notion de date est un des éléments principaux concourant au traitement de l'information. L'année 2000, codée dans ces conditions « 00 » ne peut plus être distinguée de l'année 1900. Bien sûr on peut toujours imaginer, pour certaines données, des règles logiques permettant de patienter.

À titre d'exemple, si en mars 2001, je m'intéresse à un client dont la date de naissance est « 00 » et qui est marié, je peux légitimement en déduire que ce « 00 » signifie 1900 et non pas 2000. Évidemment, cela m'oblige à coder ce test dans mon programme, donc cela ne me dispense pas d'intervenir dans le code.

Le cas énoncé est simple, mais on en comprend bien les limites, tout ne pouvant être aussi facilement résolu de cette façon. Mais ceci n'est rien, car un autre problème est l'affectation d'une clé à une information. La clé étant, chacun le sait, l'élément discriminant, jamais en double, permettant de repérer une occurrence parmi une collection d'entités. Par exemple, le numéro INSEE (plus connu sous le nom sécurité sociale) dans le fichier des assurés sociaux. On voit bien que les choses se compliquent car il y a un risque certain de répertorier des individus différents mais ayant le même numéro d'identification, ce qui n'est pas admissible dans les traitements.

Un autre inconvénient provient de la nécessité de trier les informations sur leur identifiant. Imaginons traiter les mouvements comptables d'une banque le 02 janvier 1997. Le mouvement de dépôt sur le compte en date du 31 décembre 1999 et identifié « 99 12 31 015... » sera traité **après** le mouvement de retrait du 01 janvier 2000 identifié « 00 01 01 025... » ; les conséquences sur la trésorerie des entreprises et des particuliers pouvant être catastrophiques.

Ce ne sont que quelques exemples. Il existe bien d'autres cas plus complexes où le changement de siècle induit d'autres inconvénients sur les calculs mais aussi sur les traitements d'historisation des données (les plus récentes étant épurées des fichiers avant les plus anciennes...).

Notre DI est persuadé maintenant de la gravité de la situation et de l'importance de la tâche que sa résolution implique.

Nécessité du lotissement et de l'interfaçage

Il conçoit alors que, s'il est maître de son propre système d'information, il n'en va pas de même de ceux avec lesquels il effectue des échanges : clients, fournisseurs, organismes de tutelle, État...

Il sait qu'il ne choisira pas le moment où ils seront adaptés, il subira cette transformation et son propre système d'information devra être prêt à recevoir ou à transmettre des flux aux « normes An 2000 ».

Cela induit la notion d'**interfaçage**. Les interfaces sont des couches de traitement qui s'intercalent entre un flux modifié et un traitement qui n'est pas encore prêt à traiter le 3^{ème} millénaire (ou le contraire). Il faudra donc identifier toutes les interfaces nécessaires et les concevoir. Plusieurs options sont possibles : duplication de structures de données (pour transformer l'année avant son traitement), appel à une routine de transformation dans le programme...

Un autre sujet de réflexion est la notion de lot. On ne peut traiter tous les objets composant le système d'information d'un seul coup. Il faut le découper en tranches. Chaque tranche, appelée lot, devra pouvoir être traitée indépendamment des autres lots. Cela signifie que le objets (au sens large) la composant pourront être modifiés, testés et installés en production tous ensemble.

Le **lotissement** est l'opération consistant à découper le SI en parties homogènes. Ceci n'est pas simple et requiert une bonne connaissance de l'architecture des données et des traitements ainsi que des règles de fonctionnement du SI.

On ne peut se contenter d'un découpage physique consistant à traiter dans un même ensemble tous les programmes accédant à une structure de données particulière et toutes les structures accédant à ces mêmes programmes. Se faisant, on se rend bien compte que pour les données communes (une base

comptes, une base clients, une nomenclature d'articles) un lot ainsi déterminé signifierait quasiment tout le SI. Le découpage sera donc **logique** ou ne sera pas.

Nous passerons rapidement sur le traitement technique de l'année elle-même ; plusieurs options existent (compression de la date pour ne pas agrandir les fichiers, récupération de place disponible pour agrandir la zone date, mise en œuvre d'un code du type A=19., B=20.. dans une nouvelle zone avec décodage intelligent : reconnaissance du siècle en fonction du code...). Sachez qu'il n'y a pas qu'une seule possibilité pour tout le SI, mais le choix technique sera fait pour chaque structure de données et reposera sur des critères tels que : place disponible, pérennité de l'application, etc. Ce choix est fait dès l'identification du lot, bien avant son traitement.

La problématique des tests

Modifier l'ensemble du SI implique de s'assurer que l'on n'a pas involontairement dégradé le fonctionnement antérieur des applications. Autrement dit, en plus de s'assurer que les adaptations faites pour l'An2000 fonctionnent comme prévu, il faut être certain que l'application fournira au moins le même niveau de service qu'avant les modifications. Ce type de test est appelé « **non-régression** ». Il consiste en une comparaison des résultats avant et après modification. Ces résultats, dans le cas qui nous préoccupe, devant être équivalents logiquement. Par exemple, une paie calculée avant et après transformation devra fournir les mêmes résultats. Ce type de validation est indispensable, car nul n'est à l'abri d'une modification involontaire et malencontreuse du code. Or toutes les entreprises n'effectuent pas ce genre de test, bien souvent par manque d'automatismes (et pourtant les solutions existent sur le marché).

Il sera nécessaire aussi de se projeter artificiellement dans le temps afin de valider les modifications. Ne croyons pas qu'il suffira de simuler le 01 janvier 2000, cela serait trop simple !

Selon les lots, il faudra déterminer, là encore avant le début des opérations de modification, les **dates de simulation** pertinentes (il n'y en aura pas qu'une seule). Seule la connaissance des règles de gestion permet de définir quelles sont ces dates. Du logique au physique il n'y a pas qu'un pas, car se pose la question de la résolution technique de cette simulation, mis à part un IPL (Initial Procedure Load) qui ne peut être fait à n'importe quel moment, et qui impacte l'ensemble des logiciels de base et des applicatifs. Fort heureusement, il existe des outils permettant de simuler au même moment des dates différentes pour des applications différentes.

Les clés du succès

Notre valeureux DI a maintenant pris la mesure du travail qui l'attend. Il a compris que le secret de la réussite de son projet de migration An2000 passe par trois éléments indispensables et indissociables :

- une parfaite **organisation**. Le maximum d'actions doivent être prévues le plus en amont possible du processus. Un fort niveau de coordination est nécessaire entre le traitement des différents sous-projets (un lot pouvant se traduire par un sous-projet de migration). Il lui faut donc nommer un **Directeur de Projet** ayant suffisamment d'envergure pour maîtriser un tel chantier.
- la pertinence du **lotissement**. Pas trop de lots, pas trop importants en volume, réellement homogènes. Cela passe par la maîtrise de l'architecture du SI, une **cartographie** applicative pourra beaucoup aider à la réflexion.
- l'importance du **processus de test**. Avec en prime quelques points de réflexion tels que : fera-t-on recetter les modifications à l'utilisateur final ? (Selon les sites, les habitudes ou l'approche du projet, les réponses varient.) Si la réponse est positive, le cadre de la recette doit être limité à la non-régression pour éviter une remise en cause des anciennes règles de gestion qui déboucherait sur des demandes d'évolutions fonctionnelles. Ceci passe par la mise en place d'une **stratégie de test** adaptée à l'An2000 et la mise en œuvre d'une **plate-forme de test** composée de logiciels fiables.

Ces trois éléments maîtrisés, le projet An2000 doit se dérouler sans trop de problèmes majeurs. Bien entendu, il faudra probablement faire appel à des **ressources extérieures** (on peut craindre qu'il n'y en ait pas suffisamment en interne) ; il faudra peut-être prévoir, pour certains lots, une **exploitation en double** ; il faudra se prononcer sur un **gel éventuel de l'évolution** des applications à traiter (si ce n'est pas possible, imaginer des palliatifs) ; il faudra créer un **environnement de travail** dédié (certaines entreprises réservent une partition, voire une machine au projet An2000, pour l'isoler des autres chantiers).

Un Directeur Informatique serein

À ce niveau de maturité notre DI a perçu toute la portée du projet, il sait comment le traiter, il réussira dans sa tâche.

Il y a encore quelques mois on s'interrogeait sur la **valeur ajoutée** d'un tel projet. On perçoit les éléments qui apporteront un plus à l'entreprise. Il y a bien entendu le retour d'expérience d'un projet de cet ordre (à quelques mois... ou années... du traitement de l'Euro), la constitution de la cartographie applicative permettra de mieux connaître le SI, la mise en place d'une démarche de test normée et d'un outillage adéquate fiabilisera les futurs développements et accroîtra la productivité des équipes de développement.

Est-il temps de se préoccuper du passage à l'An 2000 ?

Les grandes entreprises qui n'auront pas entamé leur projet dans les prochaines semaines (c'est-à-dire avoir au moins effectué l'analyse d'impact) auront de grandes difficultés à le mener à bien, ou alors le coût sera prohibitif, car dans quelques mois la main d'œuvre sera devenue rare.

Le poids relatif des modifications dans les traitements est peu important (maximum 15% du projet par rapport à 50% pour la phase de test) donc les automatismes à ce niveau n'offriront pas d'effets de levier remarquables. Par contre, **la phase de test, qui représente une charge de 50%** de la totalité du projet, offre des possibilités de gains de productivité très importants (un gain de 25 ou 30% sur cette phase est considérable) le retour sur investissement des acquisitions d'outils de test sera donc très rapide.

Notre DI est serein, il attend avec assurance le 01 janvier 2000. Il ne faut d'ailleurs pas se polariser sur cette date car nombre d'entreprises, si elles n'ont pas traité le sujet rapidement seront en grand danger dès 1998 ou 1999.

Conclusion

L'An 2000 est un formidable défi. Conduire un tel projet est motivant et permet d'exprimer toutes les qualités d'organisation, de rigueur, d'analyse, de management qui sont (ou devraient être) les caractéristiques premières des Chefs de Projet du futur millénaire...

Il y aurait encore beaucoup à dire, certains aspects n'ayant pu être développés dans cet article, tels que l'analyse d'impact, la nécessité de ne pas migrer des données non significatives, l'accélération du processus d'acquisition de progiciels applicatifs pour éviter la migration, et la recherche des objets concernés.

Savez-vous que certains ordinateurs personnels « perdent la tête » si on simule le 01 janvier 2000 ? Essayez de modifier la date machine de votre micro pour vous projeter au 31/12/1999 à 23h55 et observez ce qui se passe... ▲

Philippe Baschoux



Revisiter Nolan

Les étapes de la croissance informatique selon Richard Nolan

Relisons les fameuses étapes de NOLAN (1979) qui alimentaient les discours des conseils informatiques des années 80.

NOLAN considère les aspects suivants :

- *le rôle de l'utilisateur ;*
- *la planification et le contrôle ;*
- *l'organisation informatique ;*
- *le portefeuille applicatif.*

1^{ère} étape : le début

Les utilisateurs n'ont aucune prise de conscience.
Il n'y a aucune planification ni aucun contrôle informatique.
L'organisation informatique favorise l'assimilation technologique.
Les applications informatiques visent à réduire les coûts.

2^{ème} étape : la contagion

L'utilisateur manifeste un enthousiasme superficiel.
La planification et le contrôle balbutient.
L'organisation informatique tourne autour de la programmation.
Les applications prolifèrent.

3^{ème} étape : le contrôle

L'utilisateur supporte arbitrairement les responsabilités.
La planification et le contrôle sont formalisés.
L'organisation informatique s'articule autour des cadres moyens.
Les applications visent à améliorer la documentation et à restructurer les applications existantes.

4^{ème} étape : l'intégration

Les utilisateurs font l'apprentissage de la responsabilité.
On met en place un système de contrôle de planification personnalisé.
On crée des équipes centrées sur l'informatique et les comptes utilisateurs.
On utilise les techniques de rétro-conception pour rénover les applications existantes.

5^{ème} étape : la gestion des données

L'utilisateur assume une responsabilité consciente et effective.
On partage des données autour de systèmes communs.
L'organisation informatique gravite autour de la gestion des données.
Les applications sont intégrées.

6^{ème} étape : la maturité

- Les utilisateurs et le service informatique partagent les responsabilités.
- Les données constituent une ressource stratégique.
- Le système gère des bases d'informations.
- L'intégration des applications reflète les flux d'information.

Bien évidemment, en 1979, NOLAN ignorait la micro-informatique, le client-serveur, l'Internet, la gestion électronique de documents, l'infocentre qui sont venus nuancer cette belle typologie.

Mais il en reste une échelle qui permet à chacun de situer le degré de maturité informatique de son entreprise. ▲

Recueilli par Alain Coulon



Le réseau sémantique universel (1^{ère} partie)

© 1996 EPHITEQ

Des concepts semblant sans rapport entre eux présentent parfois de grandes similitudes. Voici une synthèse - très simplifiée - de vingt années de réflexion sur divers modèles de données et de traitements. L'auteur vous prie de l'excuser pour la densité - et parfois le manque d'explications ou de rigueur -, mais il est difficile de condenser en quelques pages ce qui nécessitera une publication plus complète (dans un proche avenir).

Introduction

De 1977 à 1982, j'étais, chez IBM, un des responsables du projet "Advanced Data Dictionary Utilities". Il s'agissait d'un produit maison, dont j'étais un des principaux artisans, qui transformait le dictionnaire de Données standard en un véritable AGL avant la lettre. Le support SGBD, bien que hiérarchique (DL/1), était exploité un maximum au travers de liens logiques, donnant une véritable émulation entité-relation. Malgré le grand nombre de couches logicielles (TSO / Application / Dictionnaire / IMS / DL/1 / MVS), l'importance de la mémoire consommée (de 3 à 6 Mo, valeur énorme pour l'époque) et la faible puissance - comparativement à maintenant - de la machine (un 370/168 de 1 Mips, équivalent d'un PC 286), nous parvenions à traiter 1000 - oui, mille - requêtes par minute, performance pas encore atteinte en 1996 par tous les SGBD relationnels en exploitation.

Ce niveau de performance, associé à une grande souplesse d'utilisation, m'a convaincu que l'avenir du stockage et du traitement de l'information passait par le modèle entité-relation. Des essais plus récents (1987) sur un premier prototype ont donné des résultats dépassant mes attentes (la première fois, j'ai cru que le traitement s'était interrompu prématurément) : chargement de masse d'entités avec identifiants 20 fois plus rapide qu'un simple chargement - non indexé - sous Dbase III, temps d'accès en lecture séquentielle indexée de moins d'une milliseconde sur un PC 8086, etc.

J'ai bien sûr continué d'étudier le domaine, et notamment comment intégrer un maximum d'informations dans la base afin d'automatiser au maximum les traitements (incorporation de code dans les données)¹.

Par ailleurs, il y a une dizaine d'années, je m'étais attaqué à l'élaboration du métamodèle de ce qu'on appelait alors "système expert", autrement dit un système à base de connaissances. Je cherchais notamment à permettre aisément les chaînages - avec des calculs de probabilité - tant avant qu'arrière, afin de réaliser un produit qui tournerait sur "mon" SGBD (permettant un nombre de faits et de règles illimités). Le résultat graphique de cette modélisation ressemblait beaucoup à un Modèle Conceptuel de Données, tout en fonctionnant - à la réversibilité près - comme un Modèle Conceptuel de Traitements.

Ces dernières années, j'ai donc eu la curiosité d'effectuer des comparatifs entre divers réseaux sémantiques, et j'en ai conclu qu'il devait être possible de définir une modélisation commune et de définir ainsi un système de base de données entité-relation² sur lequel on pourrait faire tourner les moteurs les plus variés.

Voici le résultat de ces cogitations.

¹ Ce modèle a d'ailleurs été étudié en 1990 par IBSI, lors de la recherche d'un SGBD pouvant accueillir la version Windows de son AGL Conceptor, et - malheureusement - non retenu.

² Ou entité-association pour ceux qui préfèrent, c'est exactement la même chose.

Les modèles analysés

Il n'est évidemment pas question d'effectuer un rapprochement entre les dizaines - sinon centaines - de sujets et de méthodes de modélisation. D'autant plus que nombre d'entre eux, s'appliquant à des domaines relativement proches, ne diffèrent que par le symbolisme et quelques détails particuliers.

Cette étude n'avait d'intérêt que si on pouvait effectivement rapprocher des systèmes a priori complètement différents, notamment en ce qui concerne leurs natures et leurs applications. Les systèmes mis en présence sont les suivants :

- Modèle Conceptuel de Données (MCD) de Merise, dans lequel on peut naviguer avec un moteur de base de données ;
- Modèle Conceptuel de Traitements (MCT) de Merise ;
- Système à Base de Connaissance (SBC), parcouru et modifié par un moteur d'inférences ;
- Tableur, mis à jour par un moteur d'algorithmes ;
- Système NeuroMimétique (SNM) ;
- Modèle de Traitements en Temps Réel (TTR) ;
- Programmation Orientée Objets (POO) telle qu'elle est mise en œuvre avec des outils comme Delphi et C++ Builder (Borland), Visual Basic et Visual C++ (Microsoft), Visual Objects (Computer Associates), NSDK (Nat Systèmes), Optima++ (Powersoft), etc.

Nous allons voir que tous ces systèmes peuvent être représentés avec un même formalisme et stockés dans un modèle unique de base de données.

Les points communs

Tous ces systèmes ont en commun qu'ils peuvent être - et sont généralement - modélisés par un réseau de nœuds reliés entre eux :

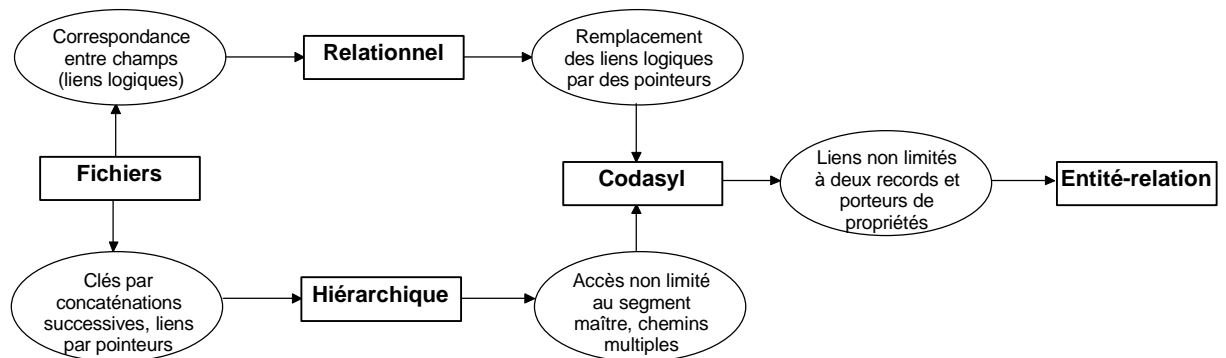
Modèle	Nœud	Lien	Particularités	Navigation
MCD	Objet	Relation	Cardinalités, contraintes ensemblistes	Base de données
MCT	Événement / Résultat	Opération	Synchronisation	Enchaînement séquentiel
SBC	Fait	Règle	Opérateurs logiques, priorités	Chaînage avant et/ou arrière
Tableur	Cellule (valeur)	Cellule (formule)	Disposition en feuilles, lignes, colonnes	Calcul et Propagation
SNM	Neurone	Axone	Synapses	Activation et Propagation
TTR	Donnée	Traitement	Capteurs et effecteurs	Algorithmes
POO	Composant / Classe	Héritage, Méthode	Méthodes incorporées aux classes	Traitement des messages

Comment nœud et lien ont-ils été distingués ?

Tout simplement selon la règle de base : un lien relie des nœuds, un nœud n'a pas nécessairement plusieurs liens :

- dans un MCT, un événement peut être initial et un résultat final, et ils ont une existence propre, alors qu'une opération se situe obligatoirement entre (au moins) un événement et (au moins) un résultat ;
- dans un SBC, un fait a une existence propre et peut être initial ou final, alors qu'une règle ne sert qu'à passer d'un fait à un autre ;
- dans un tableur, la valeur de chaque cellule a une existence propre, alors que les formules ne servent qu'à obtenir une valeur à partir d'une ou plusieurs autres ;
- dans un SNM, chaque neurone a un état spécifique, les axones (avec synapses et dendrites) ne servant qu'à les relier ;
- un TTR est analogue à un MCT ;
- dans une application objet, tout ce qui relie deux composants est un lien.

Certains pourraient critiquer que le seul représentant des SGBD pris en compte est le modèle entité-relation, et qu'il en existe d'autres. Je leur répondrai que ce modèle est le plus puissant, et que tous les autres en sont des sous-ensembles, comme le montre le diagramme ci-dessous :



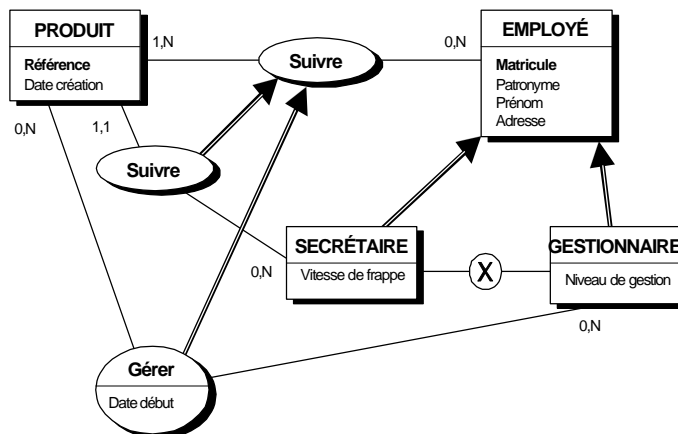
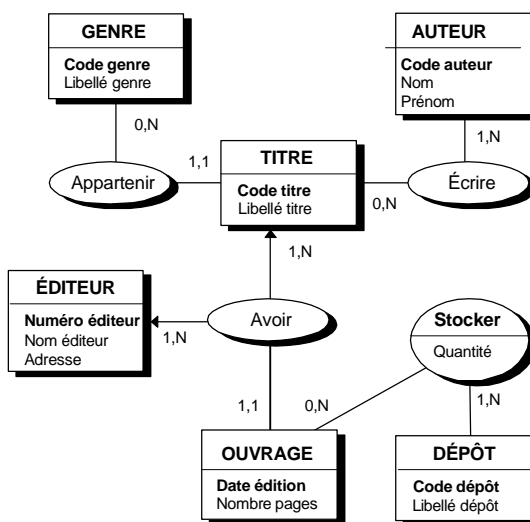
Nota : le succès actuel - et temporaire - du modèle relationnel est uniquement dû à la simplicité de sa mise en œuvre.

Les spécificités de chaque système

Le Modèle Conceptuel de Données

- Un **objet** contient de 1 à N **propriétés**, dont au moins une sert d'identifiant. Deux individus (= occurrences d'objet) ne peuvent avoir la même valeur d'identifiant.
- Une **relation** a de 2 à N **pattes**, chacune de celles-ci associant la relation à un objet (ou à une autre relation dans OOM - Orientation Objet dans Merise). À chaque patte est associée un couple de cardinalités (mini-maxi) correspondant au nombre d'occurrences de la relation pouvant être rattachées à un même individu. Une relation peut avoir des propriétés.
- Une **propriété** est une information de base. C'est une donnée indivisible dans Merise, qui peut (dans OOM) être itérative ou un agrégat (struct du C).
- Les **contraintes** ensemblistes sont un ensemble de règles définissant les possibilités d'évolution des objets et relations, que ce soit en création, mise à jour ou suppression : inclusion, exclusion, simultanéité, unicité, stabilité, etc.).
- L'**héritage** permet de définir des sous-types d'objets et de relations (généralisation par héritage simple ou multiple) et des classes (spécialisation par prédicat), en affinant les modèles avec une approche objet.

Le formalisme d'un MCD Merise est bien connu. En voici deux exemples :

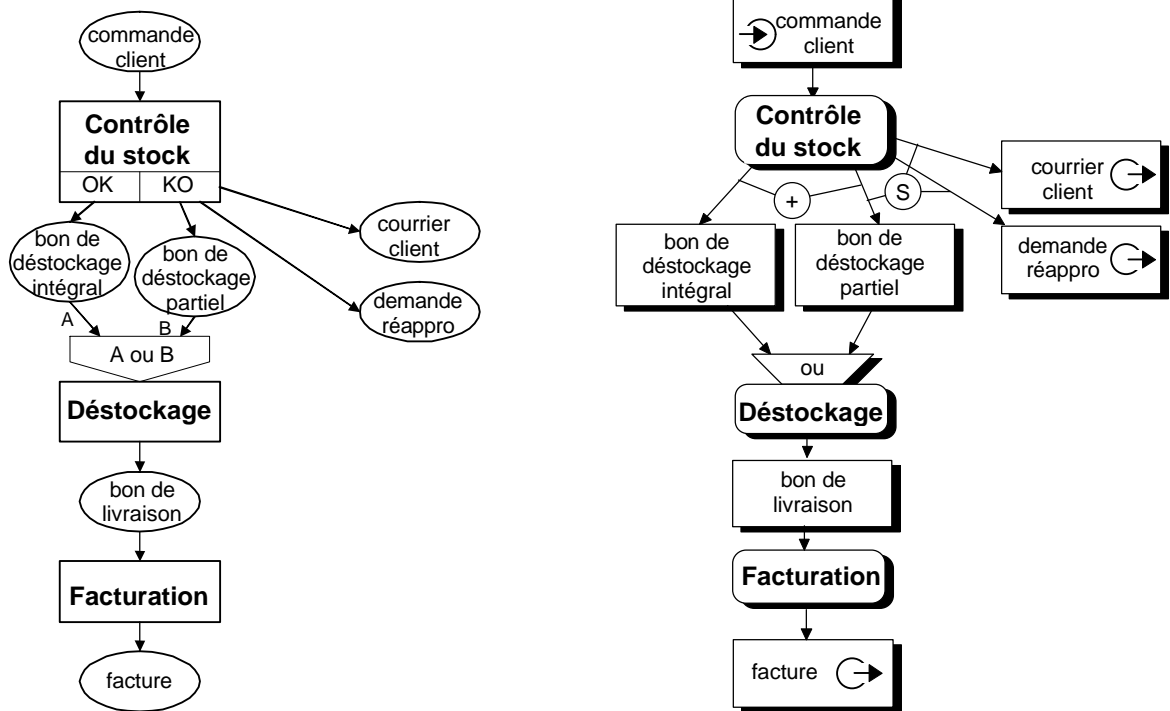


C'est ce type de formalisme - avec des enrichissements - qui sera utilisé pour l'unification des différents systèmes. Si des Merisiens puristes (dont je fus) trouvent à redire aux modèles ci-dessus, qu'ils sachent que notre objectif ici n'est pas de faire du Merise pur et dur, mais de le faire évoluer, même au prix de quelques "entorses" (c'est vrai que le suivi par la secrétaire est une relation restreinte du suivi par un employé, mais nous verrons plus loin qu'en POO le seul changement de cardinalité - qui masque un changement de méthode - correspond bien à un héritage).

Le Modèle Conceptuel de Traitements

- Un **événement** (en entrée d'une opération) ou un **résultat** (en sortie d'une opération) contient de 1 à N **propriétés**. Dans la suite, il sera sous-entendu que événement≡utilisation et résultat≡création/mise à jour.
- Une **opération** est constituée de **règles de gestion**.
- Une **synchronisation** précise les contraintes de déclenchement d'une opération.
- Une **règle de gestion** est une fonction qui utilise, crée ou modifie des propriétés d'un événement|résultat, d'un objet ou d'une relation.

Le formalisme Merise d'un MCT est également bien connu. L'exemple ci-après (à gauche) est associé au même en formalisme unifié (à droite).



Si l'intérêt de ce changement de formalisme n'apparaît pas très évident ici³, il prendra tout son sens plus loin. Bien sûr, s'agissant d'un ensemble dynamique, la signification des symboles n'est pas rigoureusement la même que dans un MCD : notamment, les "pattes" sont des chemins temporels et non spatiaux, les relations étant des actions ; par contre, les événements et résultats sont bien des objets, même s'ils n'ont qu'une durée de vie limitée.

De plus, une véritable implémentation devra descendre au niveau des règles de gestion, puisque ce sont elles qui manipulent les propriétés. Aussi, pour la suite, nous ne considérerons les opérations que comme des sous-systèmes constitués de règles de gestion reliant des objets.

Le Système à Base de Connaissances

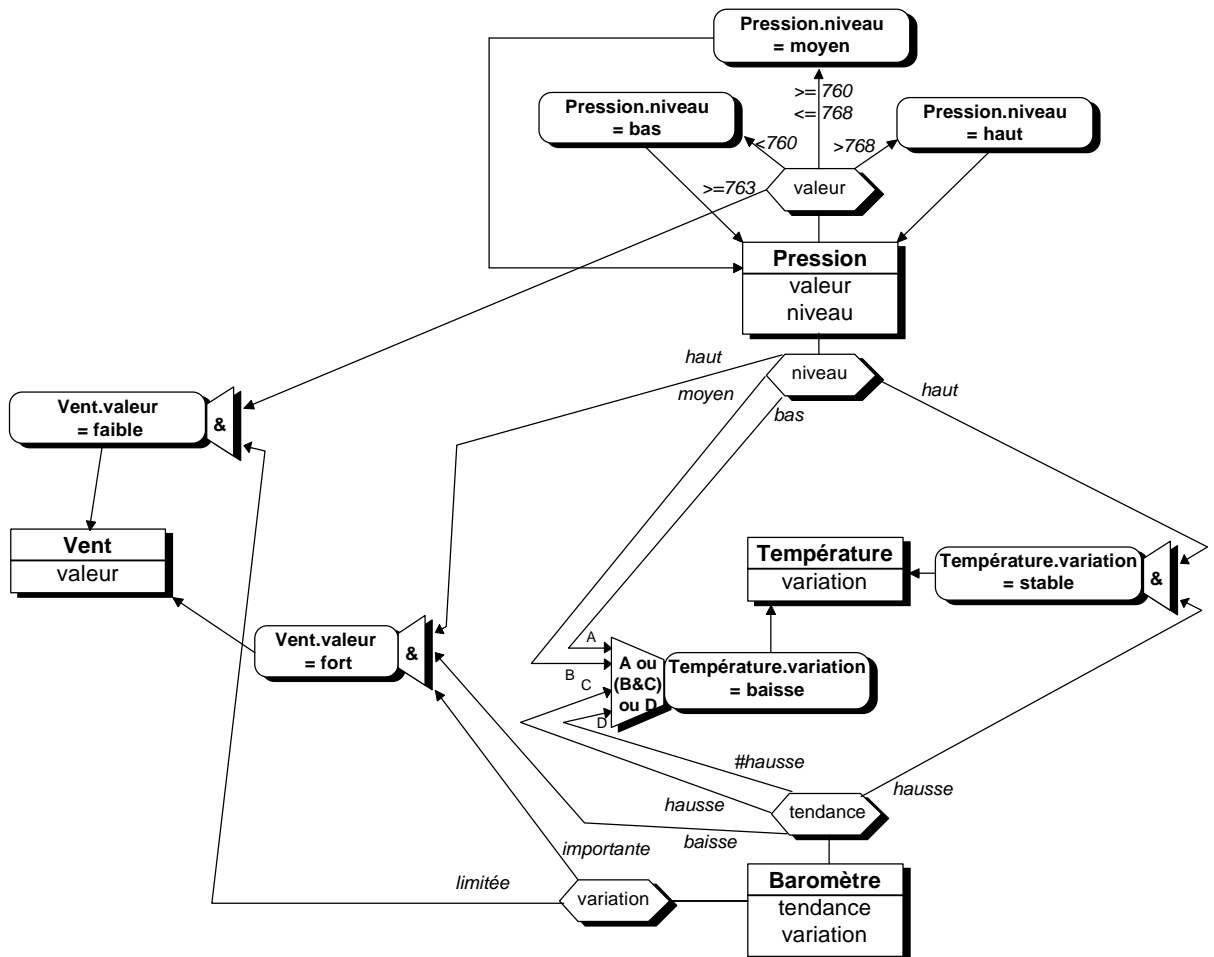
- Un **fait** est une information ayant un état ou une valeur, obtenue par l'application d'une règle.
- Une **règle** est une condition, pouvant être associée à un algorithme, qui, à partir d'une ou plusieurs conditions ou prédicats, détermine l'état ou la valeur d'un ou plusieurs faits.
- Une **prémisse** ou un **prédicat** est une condition, dont le résultat est généralement booléen.
- Une **action** est le traitement déclenché quand les prémisses qui lui sont associées sont vérifiées.
- La **priorité** permet de définir, parmi les diverses règles activables, celles qui doivent l'être en premier (celles qui sont déclenchées immédiatement - priorité la plus haute - sont parfois appelées **démons**).
- Un **chaînage avant** est un traitement qui, à partir de valeurs initiales d'un ou plusieurs faits, parcourt les règles pour en déduire l'état ou la valeur d'autres faits (raisonnement déductif).
- Un **chaînage arrière** est un traitement qui, à partir de valeurs connues d'un ou plusieurs faits, parcourt les règles pour déterminer l'état ou la valeur d'autres faits pour qu'ils soient cohérents avec les faits connus (raisonnement inductif).

³ Bien que certaines méthodes utilisent le même formalisme pour les traitements et les données ; exemple, SADT avec ses actigrammes et ses datagrammes.

De plus, il existe plusieurs niveaux de logique :

- la logique d'ordre **0**, où les faits sont booléens et uniques (Si température en hausse...);
- la logique d'ordre **0+**, où les faits peuvent constituer des populations (Si X parent de Y...);
- la logique d'ordre **1**, qui permet en plus de générer des relations entre objets
(Si terrain A au-dessus de terrain B, alors A plus récent que B...).
- la logique d'ordre **2** ? On m'en a parlé, mais je n'en ai jamais rien lu. Elle ne sera donc pas abordée ici⁴.

En ce qui concerne le formalisme, il en existe à peu près autant que de produits, aucune normalisation n'étant intervenue à ma connaissance. Voici donc un exemple - très - simple en formalisme unifié :



À noter que les objets modélisés ne représentent pas des populations, mais les individus eux-mêmes. Ceci sera explicité plus loin dans cet article⁵.

L'exemple ci-dessus n'est évidemment pas représentatif d'une modélisation de SBC en général : il faut - entre autres - différencier les prédicats simples (agissant et/ou utilisant des propriétés individuelles) des prédicats génériques (agissant sur des populations).

⁴ Toute information sur ce sujet sera la bienvenue.

⁵ Pour ceux qui remarqueraient que l'objet *Pression* n'est pas en deuxième forme normale puisque *niveau* dépend de *valeur*, je répondrai : ben oui.

Le Tableur

- Une **cellule** est l'élément de base d'un tableur. Elle ne peut contenir qu'une valeur, et est identifiée par ses coordonnées (feuille, ligne, colonne).
- Une **formule** est un algorithme, associé à la cellule qui en recevra le résultat. Les données utilisées peuvent être des constantes ou des valeurs stockées dans d'autres cellules.
- Un **calcul** est la résolution des algorithmes, qui peut se faire, soit en temps réel (au fur et à mesure des entrées de données), soit en différé (sur demande).

On retrouve ici une structure très proche d'un SBC (cellule \equiv fait, formule \equiv règle, calcul \equiv chaînage avant). Des utilitaires divers (scénarios, valeur cible...) fournissent, par approximations successives, un succédané de chaînage arrière. En fait, la principale différence entre les deux systèmes est que, dans un SBC, un fait conserve sa valeur si aucune règle ne le modifiant n'est déclenchée, alors qu'une cellule de tableur doit toujours être calculable à partir de sa formule - sauf utilisation de macros ou procédures.

La disposition en lignes et colonnes a également un rapport très proche avec les SGBD : une table relationnelle n'est pas construite autrement.

Inutile ici de montrer une modélisation de tableur, puisqu'il est possible de les utiliser tant comme SGBD que pour générer des SBC simples. D'ailleurs, compte tenu des similitudes avec les SGBD et SBC, je considérerai pour la suite qu'un tableur est une version simplifiée de l'un et/ou de l'autre, et donc que la modélisation de ceux-ci entraîne ipso facto celle des tableurs.

Le Système NeuroMimétique

- Un **neurone** est l'élément constitutif de base d'un SNM. Il ne contient - dans les modèles simples - que deux informations : son état (activé ou non) et son seuil d'activation.
- Un **axone** est le lien entre deux neurones, l'un transmetteur, l'autre récepteur.
- Une **synapse** est le point d'entrée dans un neurone depuis un axone. Elle porte une information élémentaire : activante ou inhibante.
- Un **capteur** est un point d'entrée dans un réseau de neurones. C'est un événement externe, relié à des neurones au travers d'axones et synapses.
- Un **effecteur** est un point de sortie d'un réseau de neurones. C'est un résultat externe, relié à des neurones transmetteurs.
- L'**apprentissage** est un processus de mise à niveau des seuils des neurones par itérations successives, pour obtenir en sortie le résultat attendu par rapport aux entrées (il existe aussi un mode d'apprentissage non supervisé, dans lequel on laisse le réseau s'auto-organiser).

L'ensemble **neurone.seuil + synapse** joue ici un rôle analogue à celui de la synchronisation dans un MCT.

Par ailleurs, un SNM fonctionne comme un SBC, sauf que faits (\equiv neurones) et règles (\equiv axones) sont indifférenciés et que l'ensemble peut boucler sur lui-même jusqu'à atteinte d'un état stable - ce qui existe cependant aussi dans certains SBC à système dynamique. Il peut disposer également d'un taux de bruitage (variations aléatoires du seuil d'activation) destiné à accélérer la convergence du système vers un état stable.

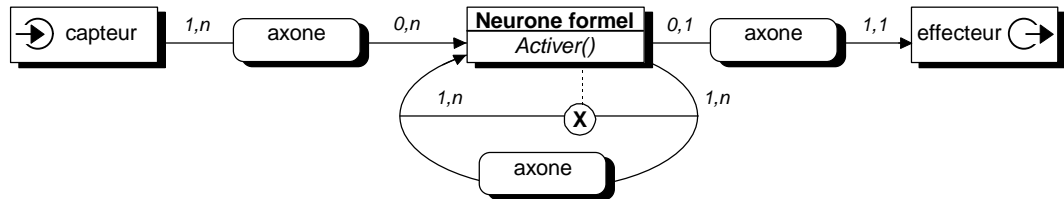
Tous les neurones étant a priori identiques, seuls comptant les informations introduites en entrée et les valeurs récupérées en sortie, une modélisation n'aurait guère de sens, sauf pour montrer le mode de couplage : rétroactivité ou non (\equiv contrainte ensembliste), niveau de couplage (\equiv cardinalité des relations).

On peut ainsi considérer un SNM comme un cas particulier de SBC à logique d'ordre 0+, la seule règle étant :

$$\text{Neurone.aktif} = (\sum \text{Valeurs(synapses)} + F(\text{bruit}) > \text{Neurone.seuil})$$

...et devant être considérée comme une méthode de l'objet Neurone (Activer()) dans l'exemple ci-dessous).

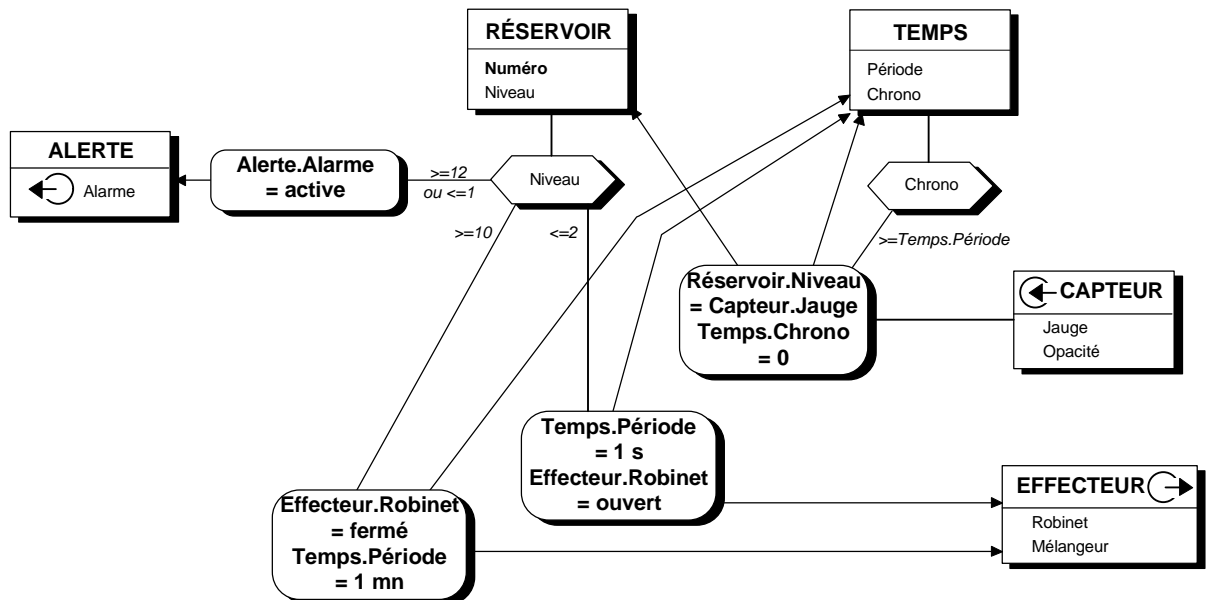
Voici, à titre d'exemple, la modélisation d'un SNM dans le cas le plus général :



Le Traitement Temps Réel

À l'exception d'une prise en compte explicite des périphériques (capteurs, effecteurs...), un Traitement Temps Réel se modélise - en formalisme unifié - exactement comme un MCT, éliminant par la même occasion l'un des reproches faits à Merise, à savoir son inadéquation au monde industriel.

En voici pour preuve un exemple de traitement - très simple -, à savoir le contrôle du niveau d'un réservoir et de son remplissage (la propriété Temps.Chrono est asservie à l'horloge et s'incrémente donc automatiquement) :



Tout comme celui du SBC, les objets du schéma ci-dessus sont directement des individus.

La Programmation Orientée Objet

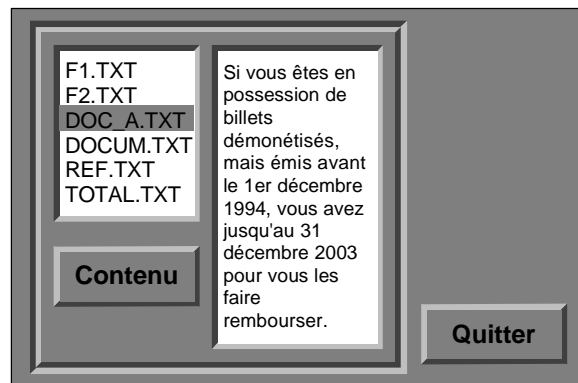
- Un **composant** (ou **objet**, ou **classe**) est, soit un élément visible et manipulable (champ de saisie, champ mémo, image, navigateur de base de données, etc.), soit un élément regroupant des traitements et des données (table relationnelle, requête SQL, etc.).
Il y a généralement une très importante arborescence d'héritage entre les divers types d'objets disponibles dans un générateur d'application.
Ainsi, tous les objets visuels ont en commun des propriétés telles que leur taille et leur position dans la fenêtre.
- Une **propriété** est une information appartenant à un composant et accessible, soit en lecture, soit en écriture, soit les deux.
Certaines propriétés cachent en fait des relations (par exemple, la propriété **Table** d'un composant **Navigateur**⁶, renseignée avec le nom de la table, génère en fait un lien entre le **navigateur** et la **table** manipulée à travers lui).
- Une **méthode** est une fonction permettant de lire ou de modifier une ou plusieurs propriétés. C'est une règle de gestion interne à l'objet.
Il y a parfois redondance entre propriété et méthode ; exemple (Delphi) : `Bouton.Hide` a exactement le même effet que `Bouton.Visible := faux`.
En réalité, les deux coexistent : la méthode `Hide` cache le bouton, puis met la propriété `Visible` à `faux`.
L'erreur du créateur de l'objet est d'avoir, dans le second cas, masqué la méthode derrière une apparente affectation de valeur : la propriété `Visible` peut être initialisée, mais ne devrait normalement être accessible à l'exécution qu'en lecture, sa mise à jour étant l'exclusivité des méthodes `Hide` et `Show`.
- Un **événement** est une action quelconque (généralement en provenance de l'utilisateur, mais parfois aussi déclenchée par une fonction) qui déclenche l'exécution d'une fonction associée. Cette fonction est toujours un lien entre deux objets (l'un d'entre eux pouvant être un événement externe au sens MCT, comme le fait de cliquer sur un bouton ou d'appuyer sur une touche de fonction).

En pratique, la mise en œuvre d'une application à l'aide de composants visuels revient plus ou moins à faire de la programmation Temps Réel, et les développeurs qui programment un bouton "Interrompre" ou une jauge permettant de visualiser l'avancement d'un traitement, ou ceux qui créent des jeux vidéo, font - dans une certaine mesure - du temps réel comme Monsieur Jourdain faisait de la prose...

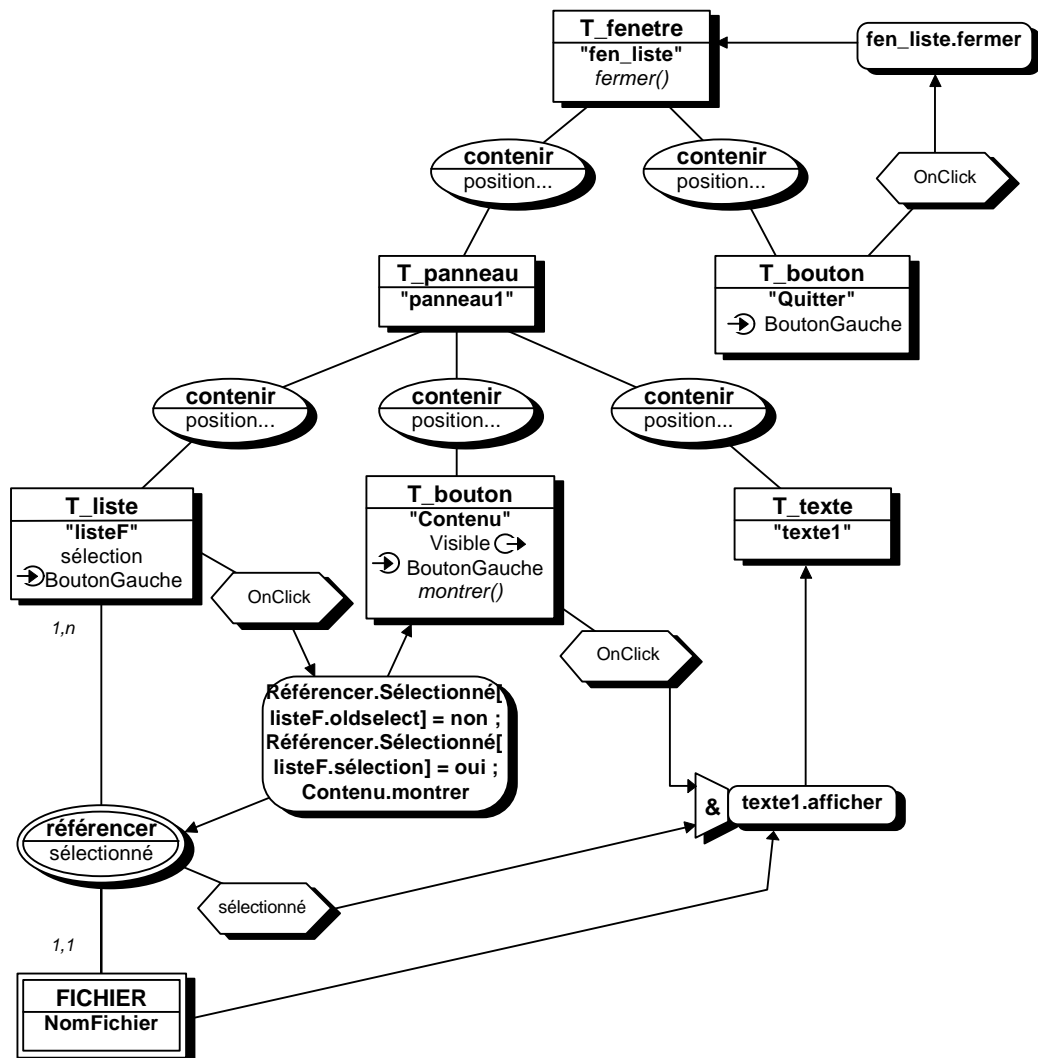
D'où l'apparente inadaptation de Merise au développement d'applications en POO.

⁶ *Navigateur* = ensemble de boutons correspondant aux fonctions premier, précédent, suivant, dernier, ajouter, supprimer, modifier, enregistrer, annuler et ayant cet aspect : 

Comment modéliser ? Eh bien, exactement comme un traitement Temps Réel : prenez l'exemple de la fenêtre ci-dessous, composée de 6 éléments : une liste (1), un texte (2), un bouton (3), les trois contenus dans un panneau (4), un autre bouton (5), l'ensemble dans la fenêtre elle-même (6).



Sachant que le bouton Contenu n'apparaît qu'après sélection d'un élément de la liste, et qu'il permet d'afficher le contenu du fichier sélectionné, on peut modéliser le tout comme suit :



Tout comme celui du SBC et du TTR, les objets du schéma ci-dessus sont directement des individus, on peut d'ailleurs le constater par la présence de deux T_bouton. Les modèles n'ont pas été représentés.

On peut remarquer les capteurs des objets listeF, Contenu et Quitter (détection des actions faites avec la souris par l'opérateur, à l'origine des événements OnClick).

Formalisme et Vocabulaire

Maintenant que nous avons vu qu'il était possible de modéliser pratiquement n'importe quoi avec un formalisme équivalent, il s'agit à présent de rassembler les différents composants de chacun de ces modèles pour élaborer un ensemble cohérent. Si le formalisme de base est celui du MCD, le formalisme unifié sera nécessairement enrichi pour prendre en compte des concepts supplémentaires⁷.

Prenons les divers concepts recensés, et voyons, catégorie par catégorie, ce qui les différencie.

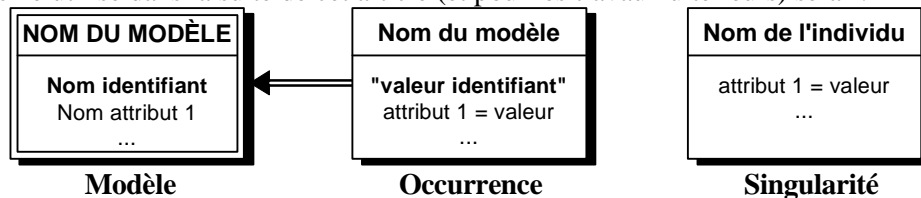
Les objets

Cette catégorie regroupe objet (pour un MCD) ; événement, résultat (pour un MCT) ; fait (pour un SBC) ; neurone, capteur, effecteur (pour un SNM). Tous sont porteurs de propriétés.

On peut diviser cet ensemble en plusieurs sous-catégories :

- Les **populations**, ensembles d'individus ayant les mêmes propriétés, différenciés par leur identifiant, et manipulés par des règles de gestion communes à tous les individus d'une même population. C'est la règle générale dans un MCD, un SNM ou un SBC d'ordre 0+. Dans une modélisation, ce type d'objet n'est représenté que par son modèle ; on ne montre pas les individus.
- Les **occurrences**, ensembles d'individus ayant les mêmes propriétés, différenciés par leur identifiant, mais pouvant être manipulés par des règles de gestion spécifiques pour chaque individu. C'est une règle courante dans un SBC (exemple : dans un ensemble de métriques, chacune - fait - est mise à jour par une ou plusieurs questions spécifiques - règles), et la généralité en POO - c'est pourquoi la modélisation nécessite de représenter les individus et non seulement les modèles.
- Les **singularités**, individus spécifiques directement identifiés par leur type (ex : SBC d'ordre 0 et/ou paramètres). Il n'y a dans ce cas évidemment aucune distinction entre le modèle et l'individu.
- Les **modèles**, qui permettent un certain niveau de généralisation : tous les individus héritent de ses attributs et relations.

Le symbolisme utilisé dans la suite de cet article (et pour les travaux ultérieurs) sera⁸ :



Les attributs

Nous parlons ici des différentes informations stockées dans les objets et dans certaines relations. Elles sont également de plusieurs types :

- Les propriétés simples, correspondant à Merise "classique", que nous appellerons désormais plutôt **attributs**, car en langage courant les propriétés d'une chose sont généralement des actions (par exemple, les propriétés d'un médicament sont des actions, alors que sa composition et son conditionnement sont des caractéristiques statiques).
- Les **faits**, ou propriétés facultatives, c'est-à-dire pouvant avoir un état non renseigné (sa valeur n'a pas été fournie), **absent** (sa valeur n'existe pas), **inconnu** (sa valeur est inconnue), **flou** (sa valeur est fournie avec une certaine probabilité d'exactitude, ou une fourchette, ou contenue dans une liste de plusieurs valeurs)⁹. Nous les identifierons en les faisant précéder du symbole : \approx . (Exemple : \approx Téléphone)

⁷ Comme le lecteur attentif s'en sera déjà aperçu dans les exemples présentés, et bien que le symbolisme proposé n'y ait pas toujours été intégralement pris en compte.

⁸ La distinction majuscules/minuscules n'est pas significative.

⁹ Par exemple, si l'attribut "n° de téléphone" n'est pas renseigné, est-ce parce qu'on ne l'a pas fourni, parce qu'on n'a pas pu l'obtenir (liste rouge...), ou parce que la personne n'a pas le téléphone ?

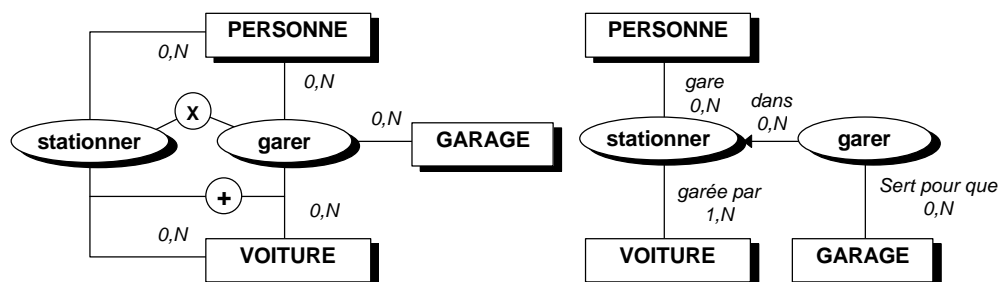
- Les **tableaux**, ou propriétés itératives (exemple : n° de téléphone). Nous les identifions en les faisant suivre d'une paire de crochets : [].
Exemple : Prénoms[]
- Les **agrégats**, propriétés décomposables en plusieurs composantes (exemple : adresse, décomposable en n°, rue, code postal, ville). On peut bien sûr avoir des agrégats contenant des tableaux, et des tableaux d'agrégats. Nous identifions un agrégat, soit en le faisant suivre de l'énumération de ses composantes encadrée par des accolades :
Exemple : Agrégat
 { attribut_a
 attribut_b
 ... }
ou, si on ne détaille pas, en le faisant suivre par : {...}.
Exemple : Agrégat {...}
- Les **méthodes**, traitements spécifiques à l'objet et permettant l'accès à un ou plusieurs attributs (en lecture et/ou en mise à jour). Elles peuvent masquer des attributs (mise à jour), ou, au contraire, faire apparaître des attributs virtuels (en lecture), valeurs renvoyées par la méthode, qui les a calculées à partir d'autres attributs. Nous les identifions en les faisant suivre d'une paire de parenthèses, contenant éventuellement la liste des paramètres transmis. Pour bien les différencier des attributs "données", il est recommandé de les mettre en italiques :
Exemples : *Cacher()*
 Afficher(oui/non)
Nota : dans un héritage, une méthode peut être redéfinie. Dans ce cas, elle remplace la méthode de l'objet parent.

Les notions de **capteur** et d'**effecteur** sont plus complexes. En effet, en POO notamment, un objet peut être à la fois capteur et effecteur (un bouton est capteur par ses méthodes générant des événements OnClick, etc.; il est effecteur par ses caractéristiques d'affichage - position, visible, etc.). Ces notions doivent donc être déportées au niveau des attributs, sauf si la totalité de l'objet est concerné - cas classique des événements/résultats externes d'un MCT. Nous identifions un capteur par le symbole : \rightarrow , et un effecteur par le symbole : \curvearrowright , placés devant le nom de l'attribut, ou devant le nom de l'objet si tous ses attributs sont des capteurs ou des effecteurs.

Les associations

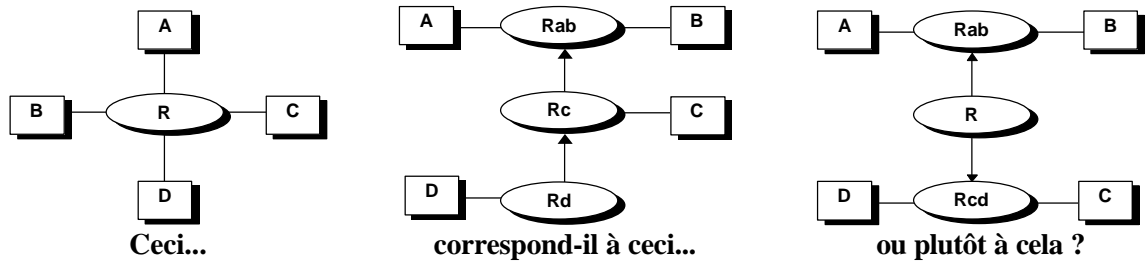
Le terme d'**association** recouvre les relations statiques, c'est-à-dire celles qui relient les objets "spatialement". Ce sont les relations classiques des MCD et SGBD.
Pour pouvoir aisément différencier les catégories - et aussi parce qu'il est extrêmement complexe, lourd et peu pratique de mettre en œuvre des relations physiques sur plus de deux objets -, nous ramènerons toutes les associations à deux pattes, grâce aux associations d'associations. En effet, dans une relation ternaire ou davantage, il est toujours possible de classer les différents objets par ordre d'importance, et donc de définir une paire qui servira d'association de base, à partir de laquelle viendront se greffer d'autres associations, jusqu'à obtention du nombre de pattes complet, ce qui peut être facteur d'économie.

Par exemple, la version de droite n'est-elle pas plus simple que celle de gauche ?



En pratique, nous verrons que comme la mise en œuvre oblige à nommer les pattes, ce nommage même permet de définir une sémantique mettant en évidence les priorités (dans l'exemple ci-dessus, on dit couramment qu'une personne gare sa voiture dans un garage, et rarement qu'une personne dans un garage gare sa voiture¹⁰; ou alors, éventuellement, qu'une personne utilise un garage pour garer sa voiture : le verbe principal définit la relation mère).

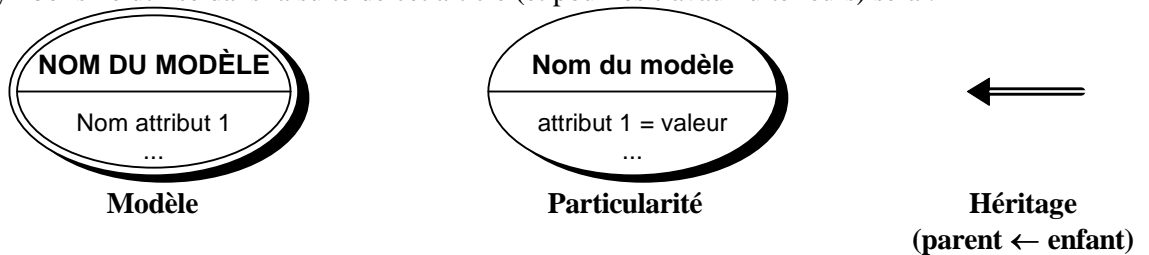
S'il y a plus de trois pattes, plusieurs possibilités existent, mais l'une d'entre elles est certainement sémantiquement meilleure :



Les différentes sous-catégories d'associations sont donc :

- Les **populations** relient deux populations d'objets, ou une population avec une occurrence ou une singularité. Dans une modélisation, ce type d'association n'est représenté que par son modèle.
- Les **particularités**, associations reliant des occurrences et/ou des singularités. Il n'y a dans ce cas pas nécessairement de modèle.
- Les **modèles**, qui permettent un certain niveau de généralisation.
- Les **héritages**, qui relient des objets (ou des associations) divers et correspondent toujours à une cardinalité (1,1) côté enfant et (0,1) côté parent :
 - * une population peut hériter d'une ou plusieurs autres populations ;
 - * une occurrence peut hériter du modèle d'une population ou d'une autre occurrence ;
 - * une singularité peut hériter d'un modèle quelconque ou d'une autre singularité.
 - * l'héritage peut être une spécialisation, auquel cas un prédicat s'interpose entre l'objet parent et l'objet enfant.

Le symbolisme utilisé dans la suite de cet article (et pour les travaux ultérieurs) sera :



Les actions

Les **actions** sont des relations "temporelles". Elles définissent - comme leur nom l'indique - des actions qui sont déclenchées lorsque les conditions définies par leurs prémisses sont remplies. Elles sont orientées de la cause vers l'effet, c'est pourquoi leurs pattes sont toujours fléchées. Elles relient des modèles, des occurrences ou des singularités ; jamais des individus de populations (ceux-ci disposent des actions de leur modèle).

Toute action est particulière : il n'y a pas de modèle d'action. Dans un "héritage", une action redéfinie remplace l'action-parent, mais ne l'enrichit pas (ni spécialisation, ni généralisation).

¹⁰ Sauf si on veut imiter Monsieur Jourdain.

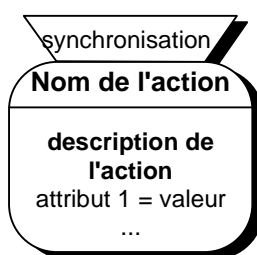
Ce qui différencie essentiellement une action d'une méthode, en dehors du fait qu'une méthode est interne à un objet - mais une action peut également ne concerner qu'un seul objet (voir par exemple `Pression.niveau` dans l'exemple de SBC présenté dans les pages précédentes) -, c'est qu'une méthode s'exécute quand elle est appelée (par une autre méthode ou par une action), alors qu'une action est déclenchée par un événement :

- externe (souris, clavier, capteur, etc.) ;
- ou interne (attribut modifié, objet ou association créé, etc.).

Ce déclenchement est soumis à un certain nombre de contraintes :

- prémisses et/ou prédicats vrais ;
- présence simultanée et/ou exclusive de prémisses et/ou prédicats, c'est-à-dire une synchronisation ;
- priorité d'exécution.

Les prédicats et prémisses sont reliés à l'action par une patte, la priorité est un attribut de l'action. La synchronisation nécessite un symbolisme particulier. Nous utiliserons donc le formalisme suivant¹¹ :

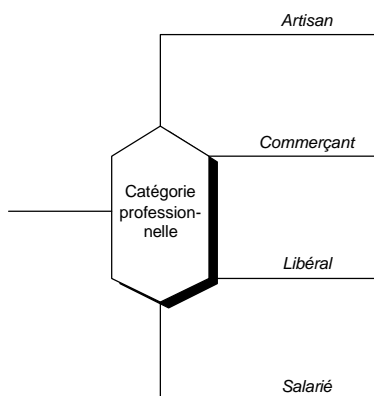


La synchronisation n'est bien sûr à représenter que si elle existe.

Les prédicats

Les **prédicats** constituent en quelque sorte une classe à part, puisqu'ils sont utilisés tant pour les données (héritage par spécialisation) que pour les actions (dans un SBC, l'ensemble prédicats+synchronisation constitue la partie si de la règle, le résultat de l'action en étant la partie alors).

Un prédicat concerne un attribut (appartenant à un objet ou à un association, ou éventuellement à une action), qui est susceptible de prendre plusieurs valeurs, ou un événement. Nous le formaliserons comme ceci :



objet ou association propriétaire de l'attribut ←

→ **objet enfant ou action utilisatrice**

À chaque patte sortante est associée la valeur du prédicat (éventuellement précédée d'un opérateur de comparaison : = # < ≤ > ≥). Cette patte est toujours fléchée, comme nous le verrons plus bas.

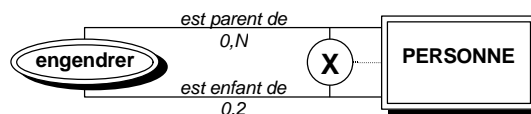
La seule patte non fléchée d'un prédicat est celle qui le relie à l'objet ou association propriétaire de l'attribut.

¹¹ Remarque aux Merisiens, qui peuvent utiliser indifféremment l'ellipse ou le rectangle arrondi pour dessiner des relations : ici, il y a une distinction fondamentale entre une association et une action : le formalisme n'est pas indifférent.

Les pattes

Les pattes sont les liens qui, dans un diagramme, relient entre eux objets, associations et actions. Recensons leurs divers types, afin de pouvoir les identifier au niveau du formalisme :

- **Objet-Objet** : il s'agit nécessairement d'un héritage. Ce type de lien est en fait une association non représentée par une ellipse, car elle n'a pas besoin de nom, et ses cardinalités sont figées (voir plus haut : associations). Sa représentation est \Longrightarrow , orientée vers l'objet parent.
- **Objet-Prédicat** : il s'agit, soit d'un prédicat sur un attribut de l'objet, auquel cas on utilise un trait simple (—), soit d'un héritage. Dans ce cas, sa représentation est \Longrightarrow , orientée vers le prédicat.
- **Objet-Association** : patte classique d'un MCD. Elle porte des cardinalités et doit être nommée avec un verbe (dont l'objet est le sujet). Ce nommage, qui peut surprendre les Merisiens (mais pas les adeptes de certaines autres méthodes) est pourtant indispensable dans le cas des relations réflexives :



Elle n'est fléchée (flèche simple \longrightarrow) que pour indiquer un identifiant relatif, et pointe dans ce cas sur l'objet portant l'identifiant principal.

- **Objet-Action** ou **Prédicat-Action** : patte indiquant un attribut utilisé ou modifié par une action. Elle est fléchée (\longrightarrow), la pointe indiquant le sens du traitement (vers l'action pour une utilisation - source -, vers l'objet pour un résultat - cible -). On peut différencier la cible en la pointant avec \longrightarrow (double flèche).
- **Association-Association** : il s'agit, soit d'un héritage, auquel cas la double flèche (\Longrightarrow) pointe vers l'association parent, soit d'une association d'associations, qu'on symbolise avec une patte simplement fléchée (\longrightarrow), la pointe indiquant l'association recevant ainsi une patte supplémentaire (facultative ou obligatoire, selon la cardinalité). Elle correspond à une identification relative, l'association source ayant besoin de l'identifiant de l'association pointée pour être complètement identifiée. Elle est nommée (terme utilisé pour construire la phrase quand on parcourt le réseau dans le sens inverse de la flèche) et porte des cardinalités.
- **Association-Action** : tout comme Objet-Action, il s'agit d'utiliser ou de modifier un attribut de l'association. Même sémantique, même formalisme (\longrightarrow ou \longrightarrow).
- **Association-Prédicat** : tout comme Objet-Prédicat, on a ici affaire à un héritage. Même sémantique, même formalisme (— depuis l'association parent, \Longrightarrow depuis l'association enfant).
- **Action-Action** : une action utilise ou modifie un attribut d'une autre action. Il est ici nécessaire de savoir si la patte indique l'utilisation par l'action cible d'un attribut de l'action source, ou de la modification d'un attribut de l'action cible par l'action source. L'option retenue est de formaliser le résultat d'une action par une patte à double flèche \longrightarrow .

Remarque importante : dans le cas d'un lien association-association correspondant à un héritage, ceci n'est valide que si l'association-enfant est reliée à un objet-enfant d'un objet-parent relié à l'association-parent (pour plus de clarté, voir le second exemple de MCD au début de l'article : Gérer \Longrightarrow Suivre vs Gestionnaire \Longrightarrow Employé).

Les contraintes

Un dernier point à aborder est celui des **contraintes** ensemblistes. Merise permet de les symboliser simplement¹²:

- Le premier groupe consiste en règles reliant des objets, des associations ou des pattes différents : inclusion $\text{---}(\text{I})\text{---}$, exclusion $\text{---}(\text{X})\text{---}$, totalité $\text{---}(\text{T})\text{---}$, partition $\text{---}(\text{+})\text{---}$, simultanéité $\text{---}(\text{S})\text{---}$, unicité $\text{---}(\text{1})\text{---}$.

¹² Ne voulant pas refaire dans cet article un cours Merise complet, je renvoie les lecteurs qui ne connaissent pas ces concepts aux sources documentaires citées ci-après.

- Le second groupe (aussi appelé contraintes de stabilité) s'applique directement à une relation, une patte ou un attribut. Le symbole se place sur la patte (relation définitive -(D) ou patte verrouillée -(V)) ou après l'attribut (propriété constante (C)).
- D'ailleurs, sémantiquement, les cardinalités sont également des contraintes ensemblistes.

Ces contraintes ensemblistes masquent en fait des méthodes et/ou des actions : quand - par exemple - on essaie de créer une troisième occurrence de relation dont la cardinalité est (0,2), il y a bien rejet quelque part, un événement erreur généré par une action, ou un code retourné par une méthode.

Ces actions, qu'on peut considérer comme banalisées (car leur algorithme n'est pas rattaché à un ensemble particulier), seront, lors d'une mise en œuvre, programmées "en dur", leur définition faisant simplement partie du descriptif du SGBD. Ceci sera étudié plus en détail dans la seconde partie de cet article.

Conclusion de la 1^{ère} partie

Nous avons donc vu que n'importe quel réseau sémantique, qu'il s'agisse de données, de traitements, d'actions... peut être modélisé dans un formalisme unique, avec cette particularité qu'on mélange parfois des entités et des modèles. C'est pourquoi la notion de métamodèle devient incontournable, et que la formalisation d'un tel réseau demande la mise en œuvre d'une nouvelle méthodologie, aucune méthode actuelle ne couvrant un champ aussi vaste¹³.

Ceci amène à une question simple : si on peut tout rassembler dans un seul modèle, n'est-il pas possible de tout réaliser avec une seule base de données ? Ma réponse est oui, et implique les contraintes et avantages suivants dans un tel système :

- Les traitements sont stockés avec les données, donc pas de programmation (ou du moins limitée au descriptif des méthodes et actions) et portabilité totale possible.
- Il est nécessaire de disposer d'un moteur de navigation (SGBD) d'une part, et d'un moteur de traitement (inférences, algorithmes) d'autre part.
- Le grand nombre d'informations modifiables simultanément et indépendamment rend obligatoire l'usage d'un véritable système d'exploitation multitâche (ce qui élimine MS-DOS, Windows 3.x et - actuellement - Apple).
- Un langage de requêtes relationnel comme SQL est absolument inadapté à un véritable système entité-relation. Un nouveau standard est donc à définir.
- Enfin¹⁴, comme il est impensable de modéliser un système en représentant l'intégralité des occurrences des entités n'appartenant pas à des populations, nous devons travailler avec un méta-modèle permettant de définir les conteneurs.

Tout ceci sera étudié dans la seconde partie, et mis en application dans un prototype qui servira à réaliser un outil d'aide au choix d'un AGL. ▲

(à suivre)

© 1996 EPHITEQ

Jean-Luc Blary

¹³ Avis aux amateurs...

¹⁴ En conclusion supplémentaire : je crains d'abuser légèrement des notes de bas de page. (Ah, vous aviez remarqué ?)

Au sommaire de la 2^{ème} partie

- ✓ Définition d'un métamodèle permettant la génération automatique
- ✓ Mise en pratique du modèle sémantique universel sous forme d'un SGBD
- ✓ Perspectives d'avenir

Sources documentaires

MERISE, support de cours
ABOUHAIR G.
IBSI, Paris 1988, 1991, 1992

De la modélisation systémique au réseau sémantique
BRES P-A., ROCHFELD A., TABOURIER Y.,
SIBERTIN-BLANC C.
Conférence-débat de l'AFCET, Paris 15/11/1990

EPHIBASE, SGBD entité-relation orienté objet,
dossiers de conception et prototype
BLARY J-L.
EPHITEQ, Caëstre 1989-1990, 1993, 1995, 1996

SGBD avancés : bases de données objet, déductives,
réparties
GARDARIN G. VALDURIEZ P.
Eyrolles, Paris 1990

ODESYS, Outil d'aide à l'évolution du Système
d'Information, dossiers de conception
BLARY J-L., THELLIEZ Ph.
EPHITEQ, Caëstre 1993-1996

Merise vers une modélisation orientée objet
MOREJON J.
Les Éditions d'Organisation, Paris 1994

MERISE & OBJETS, support de cours
BLARY J-L.
EPHITEQ, Caëstre 1995

Réseaux de neurones
NADAL J-P.
Armand Colin, Paris 1993

MERISE, support de cours
BLARY J-L.
EPHITEQ, Caëstre 1996

Moteurs de systèmes experts
VOYER R.
Eyrolles, Paris 1987

☞ *Intéressés, critiques, puristes, adversaires, partisans... pour en savoir plus ou participer :*

☎ 0.660.602.702

☎ 0.328.402.702

✉ jlblary@nordnet.fr

✉ EPHITEQ - Château Vallée - 59190 CAËSTRE



Analytique ou systémique

Les effets pervers de la comptabilité analytique

Quelques anecdotes

En 1970, je travaillais dans une société de TAF (travail à façon) qu'on appellerait aujourd'hui un centre de FM (facilities management) ou encore une infogérance. Nous gagnions modestement et honnêtement notre vie en traitant, en temps différé, les données de nos clients. Nous recevions leurs bordereaux, pleins de chiffres, et nous leur livrions des listings, pleins de colonnes.

Pour renforcer son état-major et dynamiser notre activité mécanographique, la Direction embaucha deux jeunes managers qui furent horrifiés par la rusticité de notre système de gestion.

Sur leurs conseils, on mit en place un système très perfectionné de comptabilité analytique qui mesurait les moindres flux entre les différentes unités. Le résultat ne se fit pas attendre. Deux exercices plus tard, le bilan de l'entreprise fit apparaître un déficit d'exploitation.

Ce qui fit soupirer un grincheux de la vieille école : « Avant on gagnait de l'argent, mais on ne savait pas où ; maintenant on sait exactement où en perd ! ».

Je suis très médisant, car cette opération eut, pour certains, un effet très positif. Cette société était la filiale d'un constructeur d'ordinateurs. Dans les premières vagues de démantèlements - fusions (spécialisation - généralisation), ce constructeur proposa cette filiale à une grande banque nationale.

Appliquant la même logique, le Banquier acquit la partie dont la comptabilité analytique faisait apparaître une légère marge positive et refusa de prendre en charge une activité naissante dont les investissements avaient plombé la marge globale.

Le Banquier perdit beaucoup d'argent et de crédit dans cette opération et largua ce boulet au premier repreneur venu, pour un prix symbolique. En revanche, l'activité négligée connut au sein du constructeur, qui l'avait conservée, une croissance parfaitement saine et vécut confortablement pendant une dizaine d'années.

Plus récemment, en octobre dernier, pour envoyer les exemplaires d'**ISO 9001 et développement du logiciel**, l'ADELI a fait l'emplette à La Poste de pochettes « Poste-Livre ». Pour des facilités d'horaires et de trajets, ces pochettes ont été acquises dans des bureaux de poste différents de notre bureau distributeur qui porte le nom pittoresque de Paris - Butte aux cailles.

Lors du dépôt de ce volumineux envoi postal, la Conseillère du Courrier s'est inquiétée de savoir où nous avons acheté toutes ces pochettes. Elle fut consternée par notre réponse. Le montant de 4.000 francs n'entrait pas dans ses revenus. Nous avons achevé de briser son moral, en lui révélant que nous achetions les timbres de l'ADELI, par correspondance, directement aux services philatéliques de La Poste (pour environ 40.000 francs par an).

Ainsi, nous avons découvert, avec effroi, que notre bureau distributeur supportait nos coûts sans bénéficier des revenus correspondants. À La Poste, aussi, on a réussi à établir, non pas une émulation, mais une concurrence entre les bureaux. En toute logique, il conviendrait de renforcer les bureaux où nous achetons nos enveloppes et de fermer ceux où nous les déposons.

Dans la même logique, une comptabilité analytique rigoureuse devrait fermer le rayon bricolage du sous-sol d'un grand Bazar parisien proche de l'Hôtel de Ville. Il est évident que la vente des clous, des goupilles, et autres cornières est notablement déficitaire.

La comptabilité analytique

La mise en place d'un système de comptabilité analytique consiste à disséquer l'entreprise en petites entités hiérarchisées auxquelles on affecte des numéros de sections. Il suffit ensuite de comptabiliser les flux échangés entre ces sections.

Et pour mesurer ces flux, on modélise (voir « *Ceci n'est pas une pipe* » dans *La Lettre de l'ADELI n° 24 de juillet 96*) tous les échanges en les valorisant plus ou moins arbitrairement en unités monétaires. Pour cela, il suffit de fixer et d'appliquer des barèmes de cessions des différents produits et services.

Le jeu de chaque responsable d'une section consiste à maximiser la valeur des flux qu'il émet et à minimiser la valeur de ce qu'il reçoit. Ainsi chaque section présente-t-elle un bilan satisfaisant en valorisant les moindres services qu'elle rend aux autres unités de l'entreprise et en reportant subrepticement ses coûts sur d'autres sections.

L'application de ce principe conduit à négliger tout ce qui échappe à la modélisation, en particulier, les comportements humains et les vicissitudes de l'environnement.

L'intérêt de l'entreprise n'est pas la somme des intérêts de ses composants et la Direction le comprend quand elle compare le résultat chiffré de la consolidation des résultats des sections, à la santé réelle de l'entreprise.

Et la systémique ?

L'application brutale des techniques de comptabilité analytique heurte de plein fouet les principes de la systémique.

Un système (une entreprise est un système très complexe) ne se comporte pas comme la simple addition des comportements de ses composants, mais il s'enrichit des interactions qui naissent et se développent entre ses éléments.

En outre, un système est d'autant plus efficace qu'il sait modifier sa propre structure pour mieux réagir aux perturbations de son environnement.

La comptabilité analytique fournit un modèle simple de l'entreprise. Ce modèle est facilement automatisable et fournit mathématiquement des résultats qui ne peuvent que refléter les données introduites.

Ne jetons pas le bébé avec l'eau du bain. Gardons notre comptabilité analytique ; mais sachons, tout naturellement, interpréter les résultats de cette modélisation en les replaçant dans le contexte des réalités qui, elles, échappent encore à la modélisation. ▲

Alain Coulon